Computer Sciences Department

Scheduling Data Placement Activities in Grid

Tevfik Kosar
Miron Livny

Technical Report #1483

July 2003

UNIVERSITY OF
WISCONSIN
MADISON

Scheduling Data Placement Activities in Grid

Tevfik Kosar and Miron Livny

Computer Sciences Department, University of Wisconsin-Madison
1210 West Dayton Street, Madison WI 53706
{kosart, miron}@cs.wisc.edu

Abstract

Todays scientific applications have huge data requirements, and these requirements continue to increase drastically every year. Furthermore, these data are generally accessed by many users from all across the country, or even the globe. So, there tends to be a predominant necessity to move huge amounts of data around wide area networks to complete the computation cycle, which brings with it the problem of efficient and reliable data placement. Current approach to solve this problem of data placement is either doing it manually, or employing simple scripts which do not have any automation or fault tolerance capabilities. Our goal is to make data placement activities first class citizens in the Grid just like the computational jobs. They will be queued, scheduled, monitored and managed, and even check-pointed. More importantly, it will be made sure that they complete without any human interaction.

1. Introduction

As the Grid [10] [14] evolves, the data requirements of scientific applications increase drastically. Just a couple of years ago, the data requirement for an average application was measured in Terabytes, whereas today we use Petabytes to measure it. Moreover, these data requirements continue to increase rapidly every year. A good example for this is the Compact Muon Solenoid (CMS) [6] project, a high energy physics project sponsored by the Grid Physics Network (GriPhyN) [13]. According to the Particle Physics Data Grid (PPDG) [21] deliverables to CMS, the data volume of CMS, which is currently a couple of Terabytes per year, is expected to subsequently increase rapidly, so that the accumulated data volume will reach 1 Exabyte (1 million Terabytes) by around 2015 [22]. This is the data volume required by only one application, and there are a couple of dozens of other data intensive applications with similar data needs, ranging from genomics to biomedical, and from metallurgy to cosmology.

The problem is not only the huge I/O needs of these data intensive applications, but also the number of users who will access the same datasets. For each of the projects, number of people who will be accessing the datasets range from 100s to 1000s. Furthermore, these users are not located at a single site, rather they are distributed all across the country, even the globe. So, there tend to be a predominant necessity to move huge amounts of data around wide area networks to complete the computation cycle, which brings with it the problem of efficient and reliable data placement. Data need to be located, moved, staged, replicated, and cached; storage should be allocated and de-allocated for the data whenever necessary; and everything should be cleaned up when the user is done with the data.

Just as compute resources and network resources need to be carefully scheduled and managed, the scheduling of data placement activities all across the Grid is crucial, since the access to data have the potential to become the main bottleneck for data intensive applications. Especially this is the case when most of the data is stored on tape storage systems, which slows down access to data even further due to the mechanical nature of these systems.

Currently, data placement activities in the Grid are performed either manually or by simple scripts. We can say that data placement activities are simply regarded as second class citizens of the computation dominated Grid world. Our goal is to make data placement activities first class citizens in the Grid just like the computational jobs. They need to be queued, scheduled, monitored and managed, and even check-pointed.

In the second section, we mention the challenges in the Grid to reach this goal. In the third section, we discuss the related work in this area. In the fourth section ,we introduce Stork, a prototype scheduler for data placement activities in Grid and how it can provide solutions for Grid data placement problems. In the fifth section, we give a quick view of Stork's basic components. Then we conclude with our future work and final remarks.

2. Grid Data Placement Challenges

2.1. Heterogeneous Resources

Grid is a heterogeneous environment in which many different storage systems, different data transfer middleware and protocols coexist. And it is a fundamental problem that the data required by an application might be stored in heterogeneous repositories. It is not an easy task to interact with all possible different storage systems to access the data. So there should be a unified interface using which you can access all different kind of storage systems, and also you can make use of all different underlying middleware and file transfer protocols.

2.2. Scheduling of Data Transfers

When there is a data intensive application, access to the data may become the main bottleneck in the system. Especially this is the case when the most of the data is stored on tape storage systems. The user application may not be able to access the data whenever the data is actually needed. The data should be staged from tertiary storage (tape) to secondary storage (disk), then it should be moved over the wide area network to a local storage area close to the execution site on which the application is running. If all of these are not scheduled well and the data is not made ready before hand, the application has to stay there idle for a long time waiting for the data or it will fail. Both the computational resources will be wasted and also the response time and throughput for the application will be reduced. If the data is transfered well before the application is assigned to that node, the storage resources might be held idle for a long time, not allowing other applications to use them meanwhile.

So, it should be decided well in harmony when to stage the data, when to transfer it, and when to assign the CPU on the execution site to that job. To achieve this, higher level planners and the CPU and data schedulers should work together and in harmony.

2.3. Fault Tolerance and Hiding Failures from Applications

The Grid is not a perfect environment for computation. It brings failed network connections, performance variations during transfers, crashed clients, servers and storage systems with it. But generally the applications are not prepared to these kind of problems. Most of the applications assume perfect computational environments like failure prone network and storage devices, unlimited storage, availability of the data when the computation starts, and low latency. We cannot expect every application to consider all possible failures, and performance variations in the system and be prepared for them. Instead we should be able to hide these from application by a middleware.

2.4. Different Job Requirements

Each job may have different policies, different priorities. Scheduling should be done according to the needs of each individual job. Global scheduling decisions should be able to be tailored according to the individual requirements of each job. Using only global policies may not be affective and efficient enough.

2.5. Limited Resources

The staging area or the local storage area that the application is using can be limited, which is the case generally. So the available storage area should be utilized efficiently. Suppose all of the jobs running remotely finish at the same time and try to send their outputs back to the local system. If the local storage system is not capable of storing all of the output data of those jobs, what will happen? Some of these data should be moved to other storage before the local storage are gets filled and all of the transfers fail due to lack of storage.

2.6. Resource Overloading

A common problem in distributed computing environments is that when all jobs submitted to remote sites start execution at the same time, they all start pulling data from their home storage systems (stage-in) concurrently. This can overload both network resources and the local disks of remote execution sites. It may also bring a load to the home storage systems from where the data is pulled. This problem can be easily solved by controlling the number of jobs submitted at any given time. Most job schedulers can control the number of jobs being submitted at any given time, but this solution is not sufficient always and it is not the best solution in most cases either. The reason is that it does not do any overlapping of CPU and I/O, and causes the CPU to wait while I/O is being performed. Moreover, the problem gets more complex when all jobs get completed and try to move their output data back to their home storage systems (stage-out). In this case stage-ins and stage-outs of different jobs may interfere, overloading especially the network resources more. An intelligent scheduling mechanism should be developed to control the number of stage-in and stage-outs from and to any storage system, and meanwhile do not cause any waste in CPU time.

3. Related Work

Visualization scientists at Los Alamos National Lab (LANL) found a solution for data placement by dumping data to tapes and sending them to Sandia National Laboratory (SNL) via Federal Express, because this was faster than electronically transmitting them via TCP over the 155 Mbps(OC-3) WAN backbone [8].

Reliable File Transfer Service(RFT) [19] allows byte streams to be transferred in a reliable manner. RFT can handle wide variety of problems like dropped connections, machine reboots, and temporary network outages automatically via retrying. RFT is built on top of GridFTP [1], which is a secure and reliable data transfer protocol especially developed for high-bandwidth wide-area networks.

Lightweight Data Replicator (LDR) [16] can replicate data sets to the member sites of a Virtual Organization or DataGrid. It was primarily developed for replicating LIGO [17] data, and it makes use of Globus [11] tools to replicate data. Its goal is to use the minimum collection of components necessary for fast and secure replication of data. Both RFT and LDR work only with a single data transport protocol, which is GridFTP.

There is ongoing effort to provide a unified interface to different storage systems by building Storage Resource Managers (SRMs) [25] on top of them. Currently, a couple of data storage systems, such as HPSS [24], Jasmin [4] and Enstore [9], support SRMs on top them. SRMs can also manage distributed caches using "pinning of files". The SDSC Storage Resource Broker (SRB) [2] aims to provide a uniform interface for connecting to heterogeneous data resources and accessing replicated data sets. SRB uses a Metadata Catalog (MCAT) to provide a way to access data sets and resources based on their attributes rather than their names or physical locations.

Thain et. al. propose the Ethernet approach [26] to Grid Computing, in which they introduce a simple scripting language which can handle failures in a manner similar to exceptions in some languages. The Ethernet approach is not aware of the semantics of the jobs it is running, its duty is retrying any given job for a number of times in a fault tolerant manner. Kangaroo [27] tries to achieve high throughput by making opportunistic use of disk and network resources.

Stork can use GridFTP, RFT, LDR or any other service available for reliable and secure data transfer and replication. It can make use of the services provided by SRM and SRB to access heterogeneous storage systems. It has its own fault tolerance mechanism including "retry on failure" which is the core of the Ethernet approach. One of the main goals of Stork is to work in collaboration with the on-going efforts in the data placement area and integrate or interact with them in order to provide solutions for the challenges the Grid puts in front of the users all over the globe.

```
file:/          -> local file
ftp://          -> FTP
http://         -> HTTP
gsiftp://       -> GridFTP
nest://         -> NeST (Network Storage Technologies)
srb://          -> SRB (Storage Resource Broker)
srm://          -> SRM (Storage Resorce Managers)
unitree://      -> UniTree (NCSA's Mass Storage System)
diskrouter:// -> UW DiskRouter Tool
```

Figure 1. Protocols Already Supported by Stork. *The list of data transport protocols and storage systems already supported by Stork, and how they are represented as URLs in Stork system.*

4. Stork Solutions to Grid Data Placement Problems

Stork provides solutions for many of the data placement problems encountered in the Grid environment.

4.1. Interaction with Heterogeneous Resources

Stork is completely modular, and can be extended easily. It is very straightforward to add support to Stork for your favorite storage system, data transport protocol, or middleware. This is a very crucial feature in a system which designed to work in a heterogeneous Grid environment. The users or applications may not expect all storage systems to support the same interfaces to talk to each other. And we cannot expect all applications talking to all different kinds of storage systems, protocols, and middleware. There needs to be a negotiating system between them, which can interact to those systems easily and even translate different protocols to each other. Stork has been developed to be capable of this. Modular feature of Stork allows users to insert a plugin to support their favorite storage system, protocol, or middleware easily.

Stork already has support for several different storage systems, data transport protocols, and middleware. Users can use them immediately without any extra work. The list of storage systems, protocols, and middleware that Stork can interact and the corresponding URLs that Stork uses to represent them are given in Figure 1. This list currently includes data transfer protocols such as FTP [20], GridFTP, HTTP and DiskRouter [15]; data storage systems such as SRB, UniTree [5], and NeST [3]; and data management middleware such as SRM.

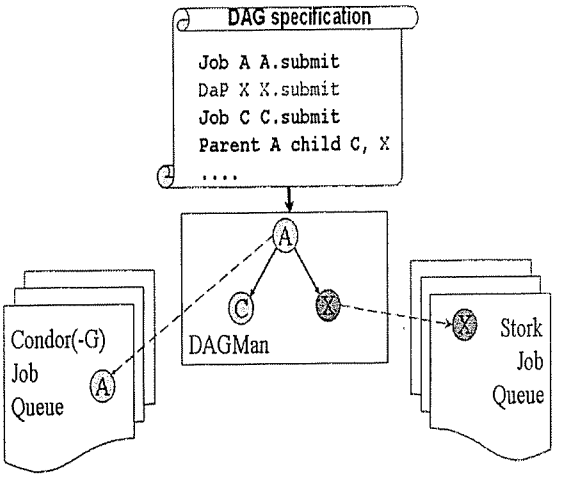

Figure 2. Interaction with Higher Level Planners. *In this prototype model, Stork interacts with DAGMan. A DAG specification file consisting of both computational and data placement jobs is submitted to DAGMan. DAGMan then submits computational jobs to Condor/Condor-G, and data placement jobs to Stork.*

4.2. Interaction with Higher Level Planners

Stork can also interact with higher level planners like DAGMan [7] [28]. This allows the users to be able to schedule both CPU resources and storage resources together. We made some enhancements to DAGMan, so that it can differentiate between computational jobs and data placement jobs. It can then submit computational jobs to a computaional job scheduler, such as Condor [18] or Condor-G [12], and the data placement jobs to Stork. Figure 2 shows a sample DAG specification file with the enhancement of data placement nodes, and how this DAG is handled by DAGMan.

In a DAG, both computational jobs and data placement jobs can be represented as nodes, and the dependencies between them can be represented as directed arcs. In this way, it can be made sure that an input file required for a computation arrives to a storage device close to the execution site, before actually that computation starts execting on that site. Similarly, the output files can be removed to a remote storage system as soon as the computation is completed. No storage device or CPU is occupied more than it is needed, and jobs do not wait idle for their input data to become available.

```
[
  dap_type    = "reserve";
  dest_host   = "db18.cs.wisc.edu";
  reserve_size = "100 MB";
  duration    = "2 hours";
  reserve_id  = 3;
]

[
  dap_type = "transfer";
  src_url  = "srb://ghidorac.sdsc.edu/home/kosart.condor/1.dat";
  dest_url = "nest://db18.cs.wisc.edu/1.dat";
]

[
  dap_type  = "release";
  dest_host = "db18.cs.wisc.edu";
  reserve_id = 3;
]
```

Figure 3. Job representation in Stork. *Three data placement requests are shown: first one to allocate space, second one to transfer a file to the reserved space, and third one to de-allocate the reserved space.*

4.3. Failure Recovery

Stork hides any kind of network, storage system, middleware, or software failures from user applications. It has a "retry" mechanism, which can retry any failing data placement job any given nuber of times before returning a failure. It has also a "kill and restart" mechanism, which allows users to specify a "maximum allowable run time" for their data placement jobs. When a job exceeds this specified time, it will be killed by Stork automatically end restarted. This will be repeated a number of times, again specified by the user. This feature provides to overcome the bugs in some systems, which causes the transfers to hang forever and never return.

4.4. Global and Job Level Policies

Stork enables users to specify job level policies as well as global ones. Global policies apply to all jobs scheduled by the same Stork server. Users can overwrite them by specifying job level policies in job description classads. The example below shows how to overwrite global policies at the job level.

```
[
  dap_type = ``transfer'';
  ...
  ...
  max_retry  = 10;
  restart_in = ``2 hours'';
]
```

In this particlar example, the user specifies that this specific job should be retried up to 10 times in case of fail-

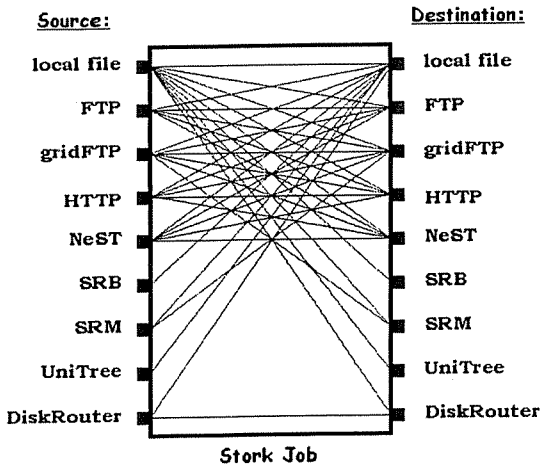

Figure 4. Protocol Translation using Stork Memory Buffer or Thirdparty Transfers. *Transfers between some storage systems and protocols can be performed directly using one Stork job via memory buffer or thirdparty transfers.*

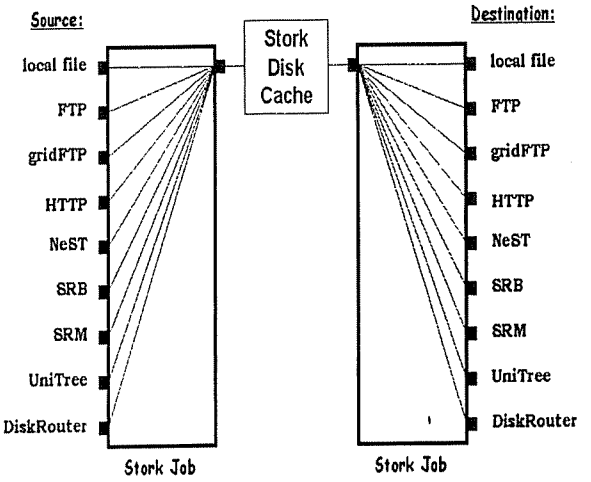

Figure 5. Protocol Translation using Stork Disk Cache. *Transfers between all storage systems and protocols supported can be performed using two Stork jobs via an intermediate disk cache.*

ures, and if the transfer does not get completed in 2 hours, it should be killed and restarted.

4.5. Efficient Resource utilization

Stork can control the number requests coming to any storage system it has access to, and makes sure that neither that storage system nor the network link to that storage system get overloaded. It can also perform space allocation and deallocations to make sure that the required storage space is available on the corresponding storage system. The space reservations are supported by Stork as long as the corresponding storage systems have support for it.

5. Stork Architecture

Two main components of Stork architecture are its flexible job representation and protocol translation features.

5.1. Job Representation

Stork uses the ClassAd [23] job description language to represent the data placement (DaP) jobs. The ClassAd language provides a very flexible and extensible data model that can be used to represent arbitrary services and constraints.

Figure 3 shows three sample data placement requests. First request is to allocate 100 MB of disk space for 2 hours on a NeST server. Second request is to transfer a file from an SRB server to the reserved space on the NeST server. And the third request is to de-allocate previously reserved space. In addition to the "reserve", "transfer", and "release", there can also be other data placement job types such as "locate", "stage" and "remove".

5.2. Protocol Translation

Stork maintains a library of pluggable "data placement" modules. These modules get executed by data placement job requests coming to Stork. They can perform inter-protocol translations either using a memory buffer or third-party transfers whenever available. Inter-protocol translations are not supported between all systems or protocols yet. Figure 4 shows the available direct inter-protocol translations that can be performed using a single Stork job.

In order to transfer data between systems for which inter-protocol translation is not supported, two consecutive Stork jobs can be used instead. First Stork job performs transfer from the source storage system to the local disk cache of Stork, and the second Stork job performs the transfer from the local disk cache of Stork to the destination storage system.

6. Future Work

Currently, the scheduling of data placement activities using Stork are performed at the file level. The users can move around only complete files. We are planning to add support for data level or block level scheduling. In this way, the

users will be able to schedule movements of partial files, or even any specific blocks of a file.

We are planning to add more intelligence and adaptation to transfers. Different data transfer protocols may have different optimum concurrency levels for any two source and destination nodes. Stork will be able to decide the concurrency level of the transfers it is performing, taking into consideration the source and destination nodes of the transfer, the link it using, and more importantly, the protocol with which it is performing the transfers. In case of availability of multiple protocols to transfer data between different nodes, Stork will be able to choose the on with the best performance, or the most reliable one according to the user preferences. And in case of a failure of a transfer due to a protocol problem, Stork will be able to try the same transfer using other protocols available.

Stork will be able to decide through which path, ideally the optimum one, to transfer data by an enhanced integration with the DiskRouter tool. It will be able to select nodes on which DiskRouters should be deployed, start DiskRouters on these nodes, and transfer the data through them by optimizing both the path and also the network utilization.

Another enhancement will be done with adding checkpointing support to data placement jobs. Whenever a transfer fails, it will not be started from scratch, but rather only the remaining parts of the file will be transfered.

7. Conclusion

We have introduced a specialized scheduler for data placement activities in Grid. Data placement efforts, which has been done either manually or by using simple scripts, are now regarded as first class citizens just like the computational jobs. They can be queued, scheduled, monitored and managed in a fault tolerant manner. We have showed the current challences with the data placement efforts in the Grid, and how Stork can provide solutions to them.

References

[1] B. Allcock, J. Bester, J. Bresnahan, A. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel, and S. T. ke. Secure, efficient data transport and replica management for high-pe rformance data-intensive computing. In *IEEE Mass Storage Conference*, San Diego, California, April 2001.

[2] C. Baru, R. Moore, A. Rajasekar, and M. Wan. The SDSC Storage Resource Broker. In *Proceedings of CASCON*, Toronto, Canada, 1998.

[3] J. Bent, V. Venkataramani, N. LeRoy, A. Roy, J. Stanley, A. C. A.-D. R. H. Arpaci-Dusseau, and M. Livny. Flexibility, manageability, and performance in a Grid storage appliance. In *Proceedings of the Eleventh IEEE Symposium on High Performance Distributed Computing (HPDC11)*, Edinburgh, Scotland, July 2002.

[4] I. Bird, B. Hess, and A. Kowalski. Building the mass storage system at Jefferson Lab. In *Proceedings of 18th IEEE Symposium on Mass Storage Systems*, San Diego, California, April 2001.

[5] M. Butler, R. Pennington, and J. A. Terstriep. Mass Storage at NCSA: SGI DMF and HP UniTree. In *Proceedings of 40th Cray User Group Conference*, 1998.

[6] CERN. The Compact Muon Solenoid Project. http://cmsinfo.cern.ch/.

[7] Condor. The Directed Acyclic Graph Manager. http://www.cs.wisc.edu/condor/dagman/, 2003.

[8] W. Feng. High Performance Transport Protocols. Los Alamos National Laboratory, 2003.

[9] FNAL. Enstore mass storage system. http://www.fnal.gov/docs/products/enstore/.

[10] I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of Supercomputing Applications*, 2001.

[11] I. Foster and C. Kesselmann. Globus: A Toolkit-Based Grid Architecture. In *The Grid: Blueprints for a New Computing Infrastructure*, pages 259–278, Morgan Kaufmann, 1999.

[12] J. Frey, T. Tannenbaum, I. Foster, and S. Tuecke. Condor-G: A Computation Management Agent for Multi-Institutional Grids. In *Proceedings of the Tenth IEEE Symposium on High Performance Distributed Computing*, San Francisco, California, August 2001.

[13] GripHyn. Grid Physics Network. http://www.griphyn.org.

[14] W. Hoschek, J. Jaen-Martinez, A. Samar, H. Stockinger, and K. Stockinger. Data Management in an International DataGrid Project. In *First IEEE/ACM Int'l Workshop on Grid Computing*, Bangalore, India, December 2000.

[15] G. Kola and M. Livny. Diskrouter: A flexible infrastructure for high performance large scale data transfers. Technical Report TR-1484, University of Wisconsin, Computer Sciences Department, 2003.

[16] S. Koranda and B. Moe. Lightweight Data Replicator. http://www.lsc-group.phys.uwm.edu/lscdatagrid/LDR/overview.html, 2003.

[17] LIGO. Laser Interferometer Gravitational Wave Observatory. http://www.ligo.caltech.edu/, 2003.

[18] M. J. Litzkow, M. Livny, and M. W. Mutka. Condor - A Hunter of Idle Workstations. In *Proceedings of the 8th International Conference of Distributed Computing Systems*, pages 104–111, 1988.

[19] R. Maddurri and B. Allcock. Reliable File Transfer Service. http://www-unix.mcs.anl.gov/ madduri/main.html, 2003.

[20] J. Postel. FTP: File Transfer Protocol Specification. RFC-765, 1980.

[21] PPDG. Particle Physics Data Grid., http://www.ppdg.net/.

[22] PPDG. PPDG Deliverables to CMS. http://www.ppdg.net/archives/ppdg/2001/doc00017.doc.

[23] R. Raman, M. Livny, and M. Solomon. Matchmaking: Distributed resource management for high throughput computing. In *Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing (HPDC7)*, Chicago, Illinois, July 1998.

[24] SDSC. High Performance Storage System (HPSS). http://www.sdsc.edu/hpss/.

[25] A. Shishani, A. Sim, and J. Gu. Storage Resource Managers: Middleware Components for Grid Storage. In *Nineteenth IEEE Symposium on Mass Storage Systems*, 2002.

[26] D. Thain, , and M. Livny. The ethernet approach to grid computing. In *Proceedings of the Twelfth IEEE Symposium on High Performance Distributed Computing*, Seattle, Washington, June 2003.

[27] D. Thain, J. Basney, S. Son, and M. Livny. The kangaroo approach to data movement on the grid. In *Proceedings of the Tenth IEEE Symposium on High Performance Distributed Computing*, San Francisco, California, August 2001.

[28] D. Thain, T. Tannenbaum, and M. Livny. Condor and the Grid. In *Grid Computing: Making the Global Infrastructure a Reality.*, Fran Berman and Geoffrey Fox and Tony Hey, editors. John Wiley and Sons Inc., 2002.